Universidad ORT Uruguay

Facultad de Ingeniería

Sistemas Operativos
Documentación

Matias Hernández - 169236

Entregado como requisito de la materia Sistemas Operativos

16 de noviembre de 2017

Resumen

Proyecto entregado con el fin de representar, probar, validar y documentar el clásico problema en los sistemas operativos : Productor-Consumidor.

El problema del Productor-Consumidor, representa al problema de sincronización de multiprocesos. En nuestro ejemplo, crearemos 2 procesos que estarán ejecutándose a la par y accederán al recurso bajo mutua exclusión.

Se darán cuatro casos de prueba para validar el programa y se concluirá que efectivamente se cumple en su totalidad la representación al problema de sincronización de multiprocesos.

Índice general

1. Introducción

Representación del problema Productor-Consumidor de los Sistemas Operativos.

Se optó por escribir un programa en el lenguaje C, usando la librería pthreads 2.9.1 para una arquitectura Windows de 32 bits.

Se utilizaron 2 thread corriendo paralelamente, uno con el proceso Productor y otro con el proceso Consumidor.

Se utilizaron 3 semáforos :

El semáforo **e** que bloquea a los productores en caso de que este lleno el buffer.

El semáforo **n** que bloquea a los consumidores en caso de que este vacío el buffer.

El semáforo **mutex** que hace mutua exclusión para acceder al buffer.

Se modificó la idea de que los productores y consumidores estén dentro de un while true do. Para realizar un ejemplo mas práctico, solamente habrán 5 productores y 5 consumidores.

El tamaño máximo de capacidad para los productos es 3, para poder bloquear a los productores cuando se llegue a esa cantidad de productos almacenados.

2. Marco teórico

2.1. Problema Productor-Consumidor

El problema Productor-Consumidor en Sistemas Operativos refleja un problema de sincronización de multiprocesos concurrentes.

Existirán diversos procesos que intenten acceder al mismo recurso; el cual deberá ser accedido bajo mutua exclusión.

Los procesos tendrán un labor, producir o consumir de dicho recurso.

Los productores, se encargaran de llenar el buffer, mientras que los consumidores, se encargarán de ir retirando del buffer.

Cuando el buffer llega a una máxima capacidad predefinida, los productores no podrán seguir insertando en el buffer. Quedarán bloqueados en espera.

Cuando el buffer esta vacío, los consumidores no podrán retirar del buffer. Quedarán bloqueados en espera.

2.2. Semáforos

Un semáforo es una variable especial (o tipo abstracto de datos) que constituye el método clásico para restringir o permitir el acceso a recursos compartidos (por ejemplo, un recurso de almacenamiento del sistema o variables del código fuente) en un entorno de multiprocesamiento (en el que se ejecutarán varios procesos concurrentemente).

Fueron inventados por Edsger Dijkstra en 1965 y se usaron por primera vez en el sistema operativo THEOS

3. Objetivo

El objetivo es establecer un contexto amigable para poder probar diferentes casos del problema de los Productores-Consumidores.

Validar que el programa cumple con el funcionamiento esperado del problema de los Productores-Consumidores.

Se quiere investigar y documentar qué pasa cuando :

Caso 1 - Los productores y consumidores, producen y consumen equitativamente.

Caso 2 - Los productores producen mucho más rápido de lo que se consumen los productos.

Caso 3 - Lo productores producen mucho más lento de lo que los consumidores consumen.

Caso 4 - Se deja aleatorio el problema, y cada uno produce y consume con un tiempo aleatorio.

Luego de investigar, concluir que la funcionalidad del programa fue correcta y que cumple con el funcionamiento esperado.

4. Código fuente

4.1. Includes

```
#define HAVE_STRUCT_TIMESPEC
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#include <windows.h>
```

4.2. Atributos Globales

```
sem_t n;
sem_t e;
sem_t mutex;

int capacidadActual = 0;
int tiempoEnProducir = 0;
int tiempoEnConsumir = 0;

const int CAPACIDAD_MAXIMA = 3;
const int CANTIDAD_PRODUCTORES = 5;
const int CANTIDAD_CONSUMIDORES = 5;
```

4.3. Main

```
int main(void)
{
  // Declaro los 2 threads que voy a usar.
  pthread_t thread0;
  pthread_t thread1;

  // Inicializo los semaforos, con valor 0.
  sem_init(&n, 0, 0);
  sem_init(&e, 0, CAPACIDAD_MAXIMA);
  sem_init(&mutex, 0, 1);

  // Se crean dos threads ejecutando el proceso
     consumidor en uno y productor en el otro.
  pthread_create(&thread0, NULL, consumidor, NULL);
  pthread_create(&thread1, NULL, productor, NULL);

  // Se unen ambos threads una vez que culminen.
  pthread_join(thread0, NULL);
  pthread_join(thread1, NULL);

  // Destruyo los semaforos.
  sem_destroy(&e);
  sem_destroy(&n);
  sem_destroy(&mutex);

  Sleep(100000);

  return 0;
};
```

4.4. Productor

```
void *productor(void *arg)
{
  for (int prodActual = 1; prodActual <=
     CANTIDAD_PRODUCTORES; prodActual++)
  {
    printf("\nProductor  %d - Produce\n", prodActual);
    // CASO 1 - Se produce equitativamente a lo que se
       cunsume.
    //tiempoEnProducir = 2000;
    // CASO 2 - Se produce mucho mas rapido que lo que se
        consume.
    //tiempoEnProducir = 2000;
    // CASO 3 - Se produce mucho mas lento que lo que se
       consume.
    //tiempoEnProducir = 6000;
    // CASO 4 - Se produce y se consume con un tiempo
       aleatorio entre 2000 y 8000.
    tiempoEnProducir = (rand() % 6001) + 2000;
    Sleep(tiempoEnProducir);

    printf("\nProductor  %d - P (e)\n", prodActual);
    sem_wait(&e);

    printf("\nProductor  %d - P (mutex)\n", prodActual);
    sem_wait(&mutex);

    printf("\nProductor  %d - Agrega producto\n",
       prodActual);
    capacidadActual++;
    printf("Cantidad de productos : %d\n",
       capacidadActual);

    printf("\nProductor  %d - V (mutex)\n", prodActual);
    sem_post(&mutex);

    printf("\nProductor  %d - V (n)\n", prodActual);
    sem_post(&n);
  }
};
```

4.5. Consumidor

```
void *consumidor(void *arg)
{
  for (int consActual = 1; consActual <=
     CANTIDAD_CONSUMIDORES; consActual++)
  {
    printf("\nConsumidor %d - P (n)\n", consActual);
    sem_wait(&n);

    printf("\nConsumidor %d - P (mutex)\n", consActual);
    sem_wait(&mutex);

    printf("\nConsumidor %d - Toma producto\n",
       consActual);
    capacidadActual--;
    printf("Cantidad de productos : %d\n",
       capacidadActual);

    printf("\nConsumidor %d - V (mutex)\n", consActual);
    sem_post(&mutex);

    printf("\nConsumidor %d - V (e)\n", consActual);
    sem_post(&e);

    printf("\nConsumidor %d - Consume\n", consActual);
    // CASO 1 - Se produce equitativamente a lo que se
       cunsume.
    //tiempoEnConsumir = 2500;
    // CASO 2 - Se produce mucho mas rapido a lo que se
       consume.
    //tiempoEnConsumir = 6500;
    // CASO 3 - Se produce mucho mas lento a lo que se
       consume.
    //tiempoEnConsumir = 2500;
    // CASO 4 - Se produce y se consume con un tiempo
       aleatorio entre 2000 y 8000 mili-segundos.
    tiempoEnConsumir = (rand() % 6001) + 2000;
    Sleep(tiempoEnConsumir);
  }
};
```

5. Dependencias adicionales

Se incluyeron pthread.h, semaphore.h, stdio.h y windows.h al proyecto.

5.1. pthread.h

POSIX Threads, usualmente llamados pthreads, is un modelo de ejecución que existe independientemente de un lenguaje, también es un modelo de ejecuciones paralelas. Permite que un programa controle múltiples flujos de trabajo que se solapan en tiempo. Cada flujo de trabajo se le llama thread y a creación y el control de dichos flujos se consigue haciendo llamadas al POSIX Threads API.

pthread_t

Es el puntero a una estructura pthread.

pthread_create

Crea un nuevo thread, con los atributos especificados en el parametro attr, dentro del proceso. Si el atributo attr es nulo, entonces se usarán los atributos default del mismo. Si el atributo especificado es modificado después, los atributos del thread no serán afectados.

pthread_join

Espera a que el thread especificado termine. Si el thread ya termino, entonces retorna inmediatamente. El thread especificado tiene que se definido como juntable.

5.2. semaphore.h

Se podrá definir el tipo sem_t, usado para ejecutar operaciones de semáforos.

sem_t

Puntero a una estructura del tipo sem.

sem_init

Inicializa un semáforo sin nombre en la dirección apuntada por sem.

sem_destroy

Destruye el semáforo en la dirección apuntada por sem.

sem_wait

Bloquea el semáforo referenciado por sem, realizando una operación de bloqueo de semáforo, en ese semáforo.

sem_post

Desbloquea el semáforo referenciado por sem, realizando una operación de desbloqueo de semáforo, en ese semáforo.

5.3. stdio.h

Standard input-output header (cabecera estándar E/S), es el archivo de cabecera que contiene las definiciones de las macros, las constantes, las declaraciones de funciones de la biblioteca estándar del lenguaje de programación C para hacer operaciones, estándar, de entrada y salida, así como la definición de tipos necesarias para dichas operaciones.

printf

Imprime la información formateada a stdout.

5.4. windows.h

Es un archivo cabecera específico de Windows para la programación en lenguaje C/C++ que contiene las declaraciones de todas las funciones de la biblioteca Windows API, todas las macros utilizadas por los programadores de aplicaciones para Windows, y todas las estructuras de datos utilizadas en gran cantidad de funciones y subsistemas.

Sleep

Suspende la ejecución del thread actual hasta que el time-out termine.

6. Pruebas

Se realizaron cuatro casos de prueba para mostrar los escenarios más relevantes del problema.

Contexto global

Cantidad de productores : 5
Cantidad de consumidores : 5
Capacidad máxima : 3 elementos

6.1. Caso 1 - Se produce equitativamente a lo que se consume

Contexto local

Tiempo en producir : 2000 ms
Tiempo en consumir : 2500 ms

Resultado

Consumidor 1 - P (n)
Productor 1 - Produce
Productor 1 - P (e)
Productor 1 - P (mutex)
Productor 1 - Agrega producto
Cantidad de productos : 1
Productor 1 - V (mutex)
Productor 1 - V (n)
Productor 2 - Produce
Consumidor 1 - P (mutex)
Consumidor 1 - Toma producto
Cantidad de productos : 0
Consumidor 1 - V (mutex)
Consumidor 1 - V (e)
Consumidor 1 - Consume
Productor 2 - P (e)
Productor 2 - P (mutex)

Productor 2 - Agrega producto
Cantidad de productos : 1
Productor 2 - V (mutex)
Productor 2 - V (n)
Productor 3 - Produce
Consumidor 2 - P (n)
Consumidor 2 - P (mutex)
Consumidor 2 - Toma producto
Cantidad de productos : 0
Consumidor 2 - V (mutex)
Consumidor 2 - V (e)
Consumidor 2 - Consume
Productor 3 - P (e)
Productor 3 - P (mutex)
Productor 3 - Agrega producto
Cantidad de productos : 1
Productor 3 - V (mutex)
Productor 3 - V (n)
Productor 4 - Produce
Consumidor 3 - P (n)
Consumidor 3 - P (mutex)
Consumidor 3 - Toma producto
Cantidad de productos : 0
Consumidor 3 - V (mutex)
Consumidor 3 - V (e)
Consumidor 3 - Consume
Productor 4 - P (e)
Productor 4 - P (mutex)
Productor 4 - Agrega producto
Cantidad de productos : 1
Productor 4 - V (mutex)
Productor 4 - V (n)
Productor 5 - Produce
Consumidor 4 - P (n)
Consumidor 4 - P (mutex)
Consumidor 4 - Toma producto
Cantidad de productos : 0
Consumidor 4 - V (mutex)
Consumidor 4 - V (e)
Consumidor 4 - Consume
Productor 5 - P (e)
Productor 5 - P (mutex)
Productor 5 - Agrega producto
Cantidad de productos : 1
Productor 5 - V (mutex)
Productor 5 - V (n)

Consumidor 5 - P (n)
Consumidor 5 - P (mutex)
Consumidor 5 - Toma producto
Cantidad de productos : 0
Consumidor 5 - V (mutex)
Consumidor 5 - V (e)
Consumidor 5 - Consume

6.2. Caso 2 - Se produce mucho más rápido a lo que se consume

Contexto local

Tiempo en producir : 2000 ms
Tiempo en consumir : 6500 ms

Resultado

Productor 1 - Produce
Consumidor 1 - P (n)
Productor 1 - P (e)
Productor 1 - P (mutex)
Productor 1 - Agrega producto
Cantidad de productos : 1
Productor 1 - V (mutex)
Productor 1 - V (n)
Consumidor 1 - P (mutex)
Productor 2 - Produce
Consumidor 1 - Toma producto
Cantidad de productos : 0
Consumidor 1 - V (mutex)
Consumidor 1 - V (e)
Consumidor 1 - Consume
Productor 2 - P (e)
Productor 2 - P (mutex)
Productor 2 - Agrega producto
Cantidad de productos : 1
Productor 2 - V (mutex)
Productor 2 - V (n)
Productor 3 - Produce
Productor 3 - P (e)
Productor 3 - P (mutex)
Productor 3 - Agrega producto
Cantidad de productos : 2
Productor 3 - V (mutex)
Productor 3 - V (n)

Productor 4 - Produce
Productor 4 - P (e)
Productor 4 - P (mutex)
Productor 4 - Agrega producto
Cantidad de productos : 3
Productor 4 - V (mutex)
Productor 4 - V (n)
Productor 5 - Produce
Consumidor 2 - P (n)
Consumidor 2 - P (mutex)
Consumidor 2 - Toma producto
Cantidad de productos : 2
Consumidor 2 - V (mutex)
Consumidor 2 - V (e)
Consumidor 2 - Consume
Productor 5 - P (e)
Productor 5 - P (mutex)
Productor 5 - Agrega producto
Cantidad de productos : 3
Productor 5 - V (mutex)
Productor 5 - V (n)
Consumidor 3 - P (n)
Consumidor 3 - P (mutex)
Consumidor 3 - Toma producto
Cantidad de productos : 2
Consumidor 3 - V (mutex)
Consumidor 3 - V (e)
Consumidor 3 - Consume
Consumidor 4 - P (n)
Consumidor 4 - P (mutex)
Consumidor 4 - Toma producto
Cantidad de productos : 1
Consumidor 4 - V (mutex)
Consumidor 4 - V (e)
Consumidor 4 - Consume
Consumidor 5 - P (n)
Consumidor 5 - P (mutex)
Consumidor 5 - Toma producto
Cantidad de productos : 0
Consumidor 5 - V (mutex)
Consumidor 5 - V (e)
Consumidor 5 - Consume

6.3. Caso 3 - Se produce mucho más lento a lo que se consume

Contexto local

Tiempo en producir : 6000 ms
Tiempo en consumir : 2500 ms

Resultado

Productor 1 - Produce
Consumidor 1 - P (n)
Productor 1 - P (e)
Productor 1 - P (mutex)
Productor 1 - Agrega producto
Cantidad de productos : 1
Productor 1 - V (mutex)
Productor 1 - V (n)
Productor 2 - Produce
Consumidor 1 - P (mutex)
Consumidor 1 - Toma producto
Cantidad de productos : 0
Consumidor 1 - V (mutex)
Consumidor 1 - V (e)
Consumidor 1 - Consume
Consumidor 2 - P (n)
Productor 2 - P (e)
Productor 2 - P (mutex)
Productor 2 - Agrega producto
Cantidad de productos : 1
Productor 2 - V (mutex)
Productor 2 - V (n)
Consumidor 2 - P (mutex)
Productor 3 - Produce
Consumidor 2 - Toma producto
Cantidad de productos : 0
Consumidor 2 - V (mutex)
Consumidor 2 - V (e)
Consumidor 2 - Consume
Consumidor 3 - P (n)
Productor 3 - P (e)
Productor 3 - P (mutex)
Productor 3 - Agrega producto
Cantidad de productos : 1
Productor 3 - V (mutex)
Productor 3 - V (n)

Consumidor 3 - P (mutex)
Productor 4 - Produce
Consumidor 3 - Toma producto
Cantidad de productos : 0
Consumidor 3 - V (mutex)
Consumidor 3 - V (e)
Consumidor 3 - Consume
Consumidor 4 - P (n)
Productor 4 - P (e)
Productor 4 - P (mutex)
Productor 4 - Agrega producto
Cantidad de productos : 1
Productor 4 - V (mutex)
Productor 4 - V (n)
Consumidor 4 - P (mutex)
Productor 5 - Produce
Consumidor 4 - Toma producto
Cantidad de productos : 0
Consumidor 4 - V (mutex)
Consumidor 4 - V (e)
Consumidor 4 - Consume
Consumidor 5 - P (n)
Productor 5 - P (e)
Productor 5 - P (mutex)
Productor 5 - Agrega producto
Cantidad de productos : 1
Productor 5 - V (mutex)
Productor 5 - V (n)
Consumidor 5 - P (mutex)
Consumidor 5 - Toma producto
Cantidad de productos : 0
Consumidor 5 - V (mutex)
Consumidor 5 - V (e)
Consumidor 5 - Consume

6.4. Caso 4 - Se produce y se consume con un tiempo aleatorio entre 2000 y 8000 mili-segundos

Contexto local

Tiempo en producir : 2000 a 8000 ms
Tiempo en consumir : 2000 a 8000 ms

Resultado

Productor 1 - Produce
Consumidor 1 - P (n)
Productor 1 - P (e)
Productor 1 - P (mutex)
Productor 1 - Agrega producto
Cantidad de productos : 1
Productor 1 - V (mutex)
Productor 1 - V (n)
Productor 2 - Produce
Consumidor 1 - P (mutex)
Consumidor 1 - Toma producto
Cantidad de productos : 0
Consumidor 1 - V (mutex)
Consumidor 1 - V (e)
Consumidor 1 - Consume
Consumidor 2 - P (n)
Productor 2 - P (e)
Productor 2 - P (mutex)
Productor 2 - Agrega producto
Cantidad de productos : 1
Productor 2 - V (mutex)
Productor 2 - V (n)
Consumidor 2 - P (mutex)
Productor 3 - Produce
Consumidor 2 - Toma producto
Cantidad de productos : 0
Consumidor 2 - V (mutex)
Consumidor 2 - V (e)
Consumidor 2 - Consume
Productor 3 - P (e)
Productor 3 - P (mutex)
Productor 3 - Agrega producto
Cantidad de productos : 1
Productor 3 - V (mutex)
Productor 3 - V (n)
Productor 4 - Produce
Consumidor 3 - P (n)
Consumidor 3 - P (mutex)
Consumidor 3 - Toma producto
Cantidad de productos : 0
Consumidor 3 - V (mutex)
Consumidor 3 - V (e)
Consumidor 3 - Consume
Consumidor 4 - P (n)
Productor 4 - P (e)

Productor 4 - P (mutex)
Productor 4 - Agrega producto
Cantidad de productos : 1
Productor 4 - V (mutex)
Productor 4 - V (n)
Consumidor 4 - P (mutex)
Productor 5 - Produce
Consumidor 4 - Toma producto
Cantidad de productos : 0
Consumidor 4 - V (mutex)
Consumidor 4 - V (e)
Consumidor 4 - Consume
Productor 5 - P (e)
Productor 5 - P (mutex)
Productor 5 - Agrega producto
Cantidad de productos : 1
Productor 5 - V (mutex)
Productor 5 - V (n)
Consumidor 5 - P (n)
Consumidor 5 - P (mutex)
Consumidor 5 - Toma producto
Cantidad de productos : 0
Consumidor 5 - V (mutex)
Consumidor 5 - V (e)
Consumidor 5 - Consume

7. Conclusión

7.1. Caso 1

Se puede apreciar que en un contexto donde el productor y el consumidor, demoran equitativamente en acceder a la zona de mutua exclusión, los productos se mantienen continuamente agregándose y quitándose del buffer.

El tiempo transcurrido para terminar con toda la operación es muy corto, ya que mientras que uno se encuentra en la zona de mutua exclusión, siempre va a haber otro, bloqueado en P (mutex), esperando a que salga el que se encuentra en la zona de mutua exclusión.

En este ejemplo se puede apreciar como se respeta la mutua exclusión a pesar de que ambos producen y consumen equitativamente.

7.2. Caso 2

Como el productor produce mucho más rápido en comparación a lo que se consume, se aprecia que en este caso, la cantidad de productos empieza a aumentar rápidamente.

Como la capacidad máxima de productos almacenados, es tres, los productores llegan a almacenar 3 productos y el próximo productor, queda bloqueado esperando a que algún consumidor, consuma un producto.

En este ejemplo se puede apreciar como funciona correctamente el bloqueo al productor cuando se llega a una capacidad máxima; y el desbloqueo del mismo cuando el siguiente consumidor, consume.

7.3. Caso 3

En este ejemplo, se puede ver como los consumidores quedan constantemente bloqueados esperando por productos para consumir, ya que los productores producen mucho más lento que la velocidad en la cual se consume.

Se puede ver como funciona correctamente la mutua exclusión y como funciona correctamente el bloqueo y el desbloqueo de los Consumidores luego de que un productor produce.

7.4. Caso 4

Este ejemplo quizás es un ejemplo mas realista, donde los productores y consumidores, producen y consumen en tiempo aleatorio.

Se puede llegar a ver que se comporta similar al Caso 1, pero esto es una coincidencia ya que el retorno mas probable, es que la cantidad de productos oscile entre 1 y 0. Pero no necesariamente tiene que ser ese el caso, en cambio en el Caso 1, si, siempre será así.

De modificarse el máximo valor del número aleatorio que se puede generar para establecer el tiempo de producción y de consumo, se podría llegar a ver que tanto los consumidores, como los productores quedaran bloqueados eventualmente. Por falta de producción o por llegar al máximo permitido de producción respectivamente.

7.5. General

Se logró establecer un contexto amigable, donde el contexto estaba bien controlado, ya que es acotado y se fue modificando dependiendo del caso que se quisiera probar.

Los cuatro casos de prueba que se establecieron, determinan que el programa se comporta correctamente siguiendo el marco teórico que representa.

Los semáforos bloquean y desbloquean correctamente y se respeta siempre la mutua exclusión gracias al uso de dichos semáforos.

Bibliografía

[1] http://man7.org/. pthread create.

[2] ——. pthread join.

[3] http://www.mit.edu/. pthread.

[4] https://www.mankier.com/. semaphore.

[5] https://linux.die.net/. sem init.

[6] http://pubs.opengroup.org/. sem wait.

[7] ——. sem post.

[8] http://www.cplusplus.com/. printf.

[9] ——. cstdio.

[10] https://es.wikipedia.org/. Windows.

[11] https://msdn.microsoft.com/. Sleep.